(c) executing said downloaded code, by an executing entity of a host system, directly from said first executable memory component, said first executable memory component being separate from said host system.

A marked-up version of the changes made to the specification and claims by the current amendment is attached, with the header "**VERSION WITH MARKINGS TO SHOW CHANGES MADE**".

REMARKS

Reconsideration of the above-identified patent application in view of the amendments above and the remarks following is respectfully requested.

Claims 1-22 are in this case. Claims 1-3, 6, 8, 10-12, 14-17 and 20-22 have been rejected under § 102(b). Claims 1-22 have been rejected under § 103(a). Claims 1-9 and 16 have been canceled. Independent claim 15 has been amended.

The claims before the Examiner are directed toward a system and method for directly executing code stored in a non-executable memory component. One or more executable memory components are provided to buffer a portion of the code to an executing entity for direct execution. When more than one executable memory component is used, then while one executable memory component functions as a buffer of one portion of the code, another portion of the code is downloaded from the non-executable memory component to another executable memory component in order to guarantee continuous access to the code by the executing entity.

§ 102(b) Rejections – Honma et al. '529

The Examiner has rejected claims 1-3, 6, 8, 10-12, 14-17 and 20-22 under § 102(b) as being anticipated by Honma et al., US Patent No. 5,606,529 (henceforth, "Honma et al. '529"). The Examiner's rejection is respectfully traversed.

Claims 1-3, 6, 8 and 16 have been canceled, thereby rendering moot the Examiner's rejection of these claims.

Honma et al. '529 teach a storage device in which data to be written to flash memories **2** or **5** by a host device or to be read from flash memories **2** or **5** by the host device is cached in a cache memory **3**. The data read by the host could be program code (column 2 lines 30-36).

The crucial difference between the present invention and the teachings of Honma et al. '529 is that the storage device of Honma et al. '529 does not present the program code to the host device in a form that is directly executable. What constitutes presentation of code to a host device in a directly executable manner is defined in the specification of the above-identified patent application on page 1 line 14 through page 2 line 2 as follows:

> The ability to execute code directly from a memory device usually requires the following characteristics from the memory device:
>
> 1. Standard memory interface, which includes address bus, data bus and a few more control signals...The executing entity...determines the address of the required code..., executes a given cycle...and expects that the required code will appear on the data bus after a short period of time...
> 2. Full visibility of the whole memory space of the memory device. This means that the executing entity can change the address values to any valid state (within the memory space of the device) and still get the required code after the same period of delay. This behavior is also known as Random Access ability. (emphasis added)

As is well-known in the art, the granularity level at which the host device retrieves code from the memory device for direct execution is very fine: typically either one byte at a time, two bytes at a time or four bytes at a time, depending on the kind of processor used by the host device. See, for example, the attached pages from *The 8086 Family User's Manual* (Intel, October 1979), Chapter 2. Note in particular the description of the operation of a microprocessor at the beginning of section 2.2:

> Microprocessors generally execute a program by repeatedly cycling through the steps shown below (this description is somewhat simplified):
> 1. Fetch the next instruction from memory.
> 2. Read an operand (if required by the instruction).
> 3. Execute the instruction.
> 4. Write the result (if required by the instruction.

and the instructions listed in Table 2-21, all of which span between 1 and 6 bytes.

Now, Honma et al. '529 state that their storage device is intended to emulate a magnetic hard disk. See, for example, column 2 lines 14-19:

> In order to provide a compatibility with a 3.5-inch, 2.5-inch or 1.8-inch hard disk device with respect to an interface at the time of data transfer, there is employed a fixed-length block format (FBA format) which is a standard format common to those devices.

and column 4 lines 61-64:

> The external specification seen from the exterior of the storage device is fully compatible with a magnetic disk device of FBA (Fixed Block Architecture) in the level of command.

To this end (column 4 line 67 through column 5 line 1),

> Data access from the host device is made in units of one block.

One FBA block typically contains 512 bytes, far more than even the largest unit (six bytes) of directly executable code. See, for example, the attached pages from *AT Attachment-3 Interface (ATA-3)* (Information Technology Industry Council, 1998). On page 1, definition 2.1.1 states the purpose of the ATA standards: "ATA defines the physical, electrical, transport, and command protocols for the internal attachment

of block storage devices" (emphasis added). On page 2, definition 2.1.15, a "sector" is defined as "a uniquely addressable set of 256 words (512 bytes)". Section 7, on pages 29-83, describes the commands of the ATA-3 standard. All of the read and write commands (7.14, READ BUFFER; 7.15, READ DMA; 7.16, READ LONG; 7.17, READ MULTIPLE, 7.18, READ SECTOR(S); 7.19, READ VERIFY; 7.34, WRITE BUFFER; 7.35, WRITE DMA; 7.36, WRITE LONG; 7.37, WRITE MULTIPLE; 7.38, WRITE SECTOR(S); 7.39, WRITE VERIFY) all read or write at least one sector. Therefore, the host device of Honma et al. '529 fails to satisfy the requirement, recited in independent claim 10 and in dependent claim 16, that the executable memory component of the present invention present the stored code in a directly executable form.

Furthermore, the present invention, as recited in independent claim 10 and in dependent claim 16, is not even obvious from Honma et al. '529. As best understood, Honma et al. '529 expect the host device to store the retrieved code in its own RAM and to execute the retrieved code directly from its own RAM. There is neither a hint nor a suggestion in Honma et al. '529 that a storage device with cached non-volatile storage can present code stored therein to a host device in directly executable form.

In response to Applicant's arguments of December 8, 2002, the Examiner denied this understanding of Honma et al. '529 on the grounds that CPU **9** of Honma et al. '529 is not shown as having direct access to any RAM other than cache memory **3**. But CPU **9** is the processor of the storage device of Honma et al. '529, not of the host device. That the only RAM directly accessible to CPU **9** is cache memory **3** is irrelevant to the host device. In any case, CPU **9** executes code for operating the storage device, not from cache memory **3**, but from ROM **10**.

While continuing to traverse the Examiner's rejections, Applicant has, in order to expedite the prosecution, chosen to amend independent claim 15 in order to clarify and emphasize the crucial distinctions between the method of the present invention and the teachings of Honma et al. '529. Specifically, independent claim 15 has been amended to include the limitation of claim 16 that the executing entity executes the downloaded code directly from the first executable memory component. Correspondingly, claim 16 has been canceled. Applicant believes that the amendment of the claims completely overcomes the Examiner's rejection of claim 15 on § 102(b) grounds.

With independent claims 10 and 15 allowable in their present form, it follows that claims 11, 12, 14, 17 and 20-22, that depend therefrom, also are allowable.

§ 103(a) Rejections – Pashley et al. '506 in view of Garvin et al. '156 or Alexis et al. '103

The Examiner has rejected claims 1-6, 8, 10, 11, 14-17 and 20-22 under § 103(a) as being unpatentable over Pashley et al., US Patent No. 6,418,506 (henceforth, "Pashley et al. '506") in view of Garvin et al., US Patent No. 6,260,156 (henceforth, "Garvin et al. '156") or Alexis et al., US Patent No. 6,260,103 (henceforth, "Alexis et al. "103"). The Examiner's rejection is respectfully traversed.

Claims 1-6, 8 and 16 have been canceled, thereby rendering moot the Examiner's rejection of these claims.

Pashley et al. '506 teach a memory device **100** in which data to be written to a flash array (identified by reference numeral **103** in Figure 1 and by reference numeral **203** in Figure 2) or read from flash array **103/203** by a processor (identified by reference numeral **104** in Figure 1 and by reference numeral **200** in Figure 2) are cached in a RAM write buffer array (identified by reference numeral **101** in Figure 1

and by reference numeral **202** in Figure 2). In a read operation, as described with reference to Figure 2, requested data is identified by an address **201** that is sufficiently detailed to identify a single word in either RAM write buffer array **101/202** or flash array **103/203**.

Pashley et al. '506 is silent concerning whether the data stored in flash array **103/203** could be executable code. However, as correctly pointed out by the Examiner, both Garvin et al. '156 (column 4 line 38) and Alexis et al. '103 (column 1 line 30) teach that executable code can be stored in a flash memory. Ostensibly, it would follow that the present invention is an obvious combination of the teachings of Pashley et al. '506 and either Garvin et al. '156 and Alexis et al. '103.

The flaw in this reasoning is that the present invention, as recited in claims 10 and 15, stores the executable code in a non-executable memory. The specification of the above-identified patent application defines a non-executable memory, on page 2 lines 3-4, as a memory that does not satisfy the requirements set forth on page 1 line 14 through page 2 line 2, and cited above, for supporting direct execution of code. By contrast, the flash memories of Pashley et al. '506, Garvin et al. '156 and Alexis et al. '103 all are NOR flash memories, which, as random access memories, do support direct execution of code and so are executable memory components in the terminology of the above-identified patent application.

It is well-known in the art that NOR flash memories and NAND flash memories have quite different access and timing characteristics. NOR flash memories are random access devices that have the same read time as RAM (several tens of nanoseconds), write times on the order of microseconds and erase times on the order of hundreds of milliseconds. NAND flash memories are non-random-access devices that have read times of 10 to 15 microseconds, write times of hundreds of

microseconds and erase times of milliseconds. Attached please find a partial copy of a datasheet of an exemplary NOR flash memory, specifically the Intel 28F008SA, dated December 1998, and a partial copy of a datasheet of an exemplary NAND flash memory, specifically the Toshiba TC58V32ADC, dated 7 November 1998. The first paragraph of the Intel datasheet defines the Intel device as an "eXecute-In-Place (XIP) device, *i.e.*, what the above-identified patent application calls an "executable memory component". The read time of the Intel device is the "Read Cycle Time" on page 29: 85 to 120 nanoseconds. The write time of the Intel device is the "Duration of Byte Write Operation" on page 33: 6 microseconds. The erase time of the Intel device is the "Duration of Block Erase Operation" on page 33: 300 milliseconds. The first paragraph of the Toshiba datasheet defines the Toshiba device as a NAND EEPROM. The second paragraph of the Toshiba datasheet defines the Toshiba device as a "serial type of memory device", *i.e.*, not a random access device. The read time of the Toshiba device is "t_R" on page 4: 10 microseconds. The write time of the Toshiba device is the "Average Programming Time" on page 5: 300 to 1000 microseconds. The erase time of the Toshiba device is the "Block Erasing Time" on page 5: 2 to 30 milliseconds.

That flash array **103/203** of Pashley et al. '506 is a NOR flash array is clear from column 2 lines 17-19:

> Data can be very quickly read from a flash memory device, nearly as fast or faster than the same data can be read from a RAM device.

That the flash memory of Garvin et al. '156 is a NOR flash memory is clear from column 1 lines 46-48:

> The erasing of an entire block may take from several tens of milliseconds to several seconds.

That the flash memory of Alexis et al. '103 is a NOR flash memory is clear from column 1 lines 41-45:

> For example, in some current flash memories, a read operation may take approximately 100 nanoseconds to perform, while a program operation may require about 10 microseconds and an erase cycle may take as long as one second to perform.

Note that Alexis et al. '103 use "program" as a synonym for "write (column 1 lines 36-37).

Further evidence that flash array **103/203** of Pashley et al. '506 is a random access memory is found in Figure 2 and the description thereof in column 5 line 19 through column 6 line 43. Specifically, the process of reading data that may be stored either in flash array **103/203** or in RAM write buffer array **101/202** includes the following operations:

> Column 6 lines 15-17:
>
> Responsive to the block offset value of address **201**, one of the four 64 bit data elements i, xb, xc or xd is selected and passed on to output line **210**.
>
> Column 6 lines 29-32:
>
> Output line **210** is applied to the input of multiplexer **207**. Also applied to the input of multiplexer **207** is the requested data from flash array **203** at the appropriate address location y.

Pashley et al. '506 do not say explicitly that the input from flash array **103/203** to multiplexer **207** is a single 64 bit data element, but if the input from RAM write buffer array **101/202** to multiplexer **207** is a single 64 bit data element then the input from flash array **103/203** to multiplexer **207** must also be a single 64 bit data element. If only a single 64 bit data element, and not just a whole block of data elements, can be read directly from flash array **103/203**, then flash array **103/203** must be a random access memory.

It follows that the obvious combination of Pashley et al. '506 with either Garvin et al. '156 or Alexis et al. '103 is not a non-executable memory component (*e.g.*, NAND flash) buffered for direct execution via an executable memory component, but a non-volatile executable memory component (*e.g.*, NOR flash) buffered for execution via a volatile executable memory component.

There is a further reason why one ordinarily skilled in the art would not consider basing a directly executable memory device on the teachings of Pashley et al. '506, with or without Garvin et al. '156 and Alexis et al. '103. Recall that one of the requirements for a directly executable memory device, as defined in the above citation from page 1 line 14 through page 2 line 2 of the above-identified patent application, is "full visibility", meaning that the required code is retrieved from any address after substantially the same delay. Now, in the flow chart of Figure 4 of Pashley et al. '506 (note that "REAL REQUEST" in block **405** should be "READ REQUEST"), one of the contingencies of a read request is having to wait in block **460** for the merge buffer (*i.e.*, RAM write buffer array **202**: see column 6 lines 48-53) to empty. Because emptying the merge buffer includes writing to flash array **203**, emptying the merge buffer is a slow operation. Consequently, if memory device **100** were used to present code to processor **104/200** for direct execution, processor **104/200** sometimes would have to wait an unknown and variable amount of time for items of code from memory device **100**.

It follows that the present invention, as recited in independent claim 10 and in independent claim 15 as now amended, is patentable over the combination of Pashley et al. '506 and either Garvin et al. '156 or Alexis et al. '103. It follows further that claims 11, 14, 17 and 20-22, that depend therefrom, also are allowable.

§ 103(a) Rejections – Nojima '634 in view of Garvin et al. '156 or Alexis et al. '103

The Examiner has rejected claims 1-22 under § 103(a) as being unpatentable over Nojima, US Patent No. 6,246,634 (henceforth, "Nojima '634") in view of either Garvin et al. '156 or Alexis et al. '103. The Examiner's rejection is respectfully traversed.

Claims 1-9 and 16 now have been canceled, thereby rendering moot the Examiner's rejection of these claims.

Nojima '634 teaches a memory circuit in which (in the "second mode of operation" described in column 3 line 44 through column 4 line 45) data to be written to one of flash memory arrays **12** and **16** may first be written to one of SRAM memory arrays **14** and **18** and then copied to the associated flash memory array **12** or **16**, and in which data to be read from one of flash memory arrays **12** and **16** may first be copied to the associated SRAM memory array **14** or **18** and then read from that SRAM memory array **14** or **18**. Like Pashley et al. '506, Nojima '634 is silent concerning whether the data stored in flash memory arrays **12** and **16** could be executable code. Ostensibly, as in the case of Pashley et al. '506, one ordinarily skilled in the art could learn from either Garvin et al. '156 or Alexis et al. '103 that the data stored in flash memory arrays **12** and **16** of Nojima '634 could include executable code, thereby rendering the present invention obvious.

As in the case of Pashley et al. '506, the flaw in this reasoning is that the present invention, as recited in claims 10 and 15, stores the executable code in a non-executable memory. It is clear from Nojima '634 that flash memory arrays **12** and **16** are random access memory arrays (presumably, NOR flash arrays). First, flash memory arrays **12** and **16** share common address buses **24** and **26** and common data

buses **20** and **22** with SRAM memory arrays **14** and **18** (which, of course, are random access memory arrays). Second, the transfer commands in column 3 line 64 through column 4 line 13 for copying between flash memory arrays **12** and **16** and SRAM memory arrays **14** and **18** include byte mode commands, implying that individual bytes of flash memory arrays **12** and **16**, and not just whole sectors of flash memory arrays **12** and **16**, can be addressed for reading and writing. Third, in the embodiment of Figure 2, flash memory arrays **12** and **16** are treated equivalently to SRAM memory arrays **14** and **18**:

> Column 5 lines 25-27:
>
> The output of the second multiplexer **82** is connected to the data port of the memory array (**12**, **14**, **16** or **18**).
>
> Column 5 lines 27-29:
>
> ...the data signals from the memory array (**12**, **14**, **16** or **18**) is provided to a demultiplexer **84**...

implying that data of the same size (bytes, not sectors) are read from and written to both SRAM memory arrays **14** and **18** and flash memory arrays **12** and **16**. It follows that, as in the case of Pashley et al. '506, the obvious combination of Nojima '634 with either Garvin et al. '156 or Alexis et al. '103 is not a non-executable memory component (*e.g.*, NAND flash) buffered for direct execution via an executable memory component, but a non-volatile executable memory component (*e.g.*, NOR flash) buffered for execution via a volatile executable memory component. It then follows that the present invention, as recited in independent claim 10 and in independent claim 15 as now amended, is patentable over the combination of Nojima '634 and either Garvin et al. '156 or Alexis et al. '103.

With independent claims 10 and 15 allowable in their present form, it follows that claims 11-14 and 17-22, that depend therefrom, also are allowable.

§ 103(a) Rejections – Honma et al. '529, or Pashley et al. '506 in view of Garvin et al. '156, or Pashley et al. '506 in view of Alexis et al. '103, and further in view of Shimizu '535, Kikuchi '632, Claxton '293 and Sakamoto '201

The Examiner has rejected claims 7, 18 and 19 under § 103(a) as being unpatentable over either Honma et al. '529 or Pashley et al. '506 in view of Garvin et al. '156 or Pashley et al. '506 in view of Alexis et al. '103, all further in view of Shimizu, US Patent No. 5,644,535, Kikuchi, US Patent No. 6,477,632, Claxton, US Patent No. 5,901,293 and Sakamoto, US Patent No. 5,303,201. The Examiner's rejection is respectfully traversed.

Claim 7 has been canceled, thereby rendering moot the Examiner's rejection of this claim.

It is demonstrated above that independent claim 15 is allowable in its present form. It follows that claims 18 and 19, that depend therefrom, also are allowable.

In view of the above amendments and remarks it is respectfully submitted that independent claims 10 and 15, and hence dependent claims 11-14 and 17-22 are in condition for allowance. Prompt notice of allowance is respectfully and earnestly solicited.

Respectfully submitted,

Mark M. Friedman
Attorney for Applicant
Registration No. 33,883

Date: May 8, 2003

VERSION WITH MARKINGS TO SHOW CHANGES MADE

In the claims:

Claim 15 has been amended as follows:

15. (Amended) A method of executing code, comprising the steps of:

(a) storing the code in a non-executable memory component;

(b) downloading at least a portion of the code from said non-executable memory component to a first executable memory component; and

(c) executing said downloaded code, by an executing entity of a host system, directly from said first executable memory component, said first executable memory component being separate from said host system.